

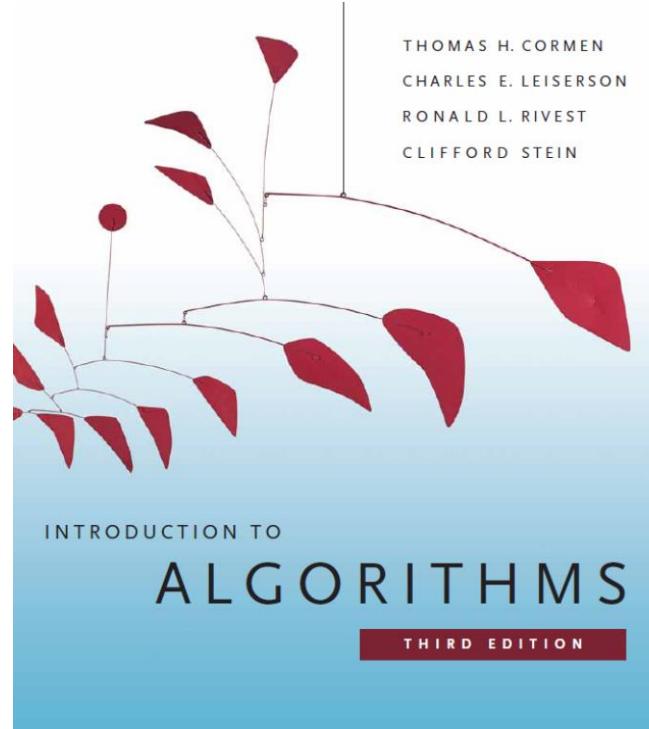
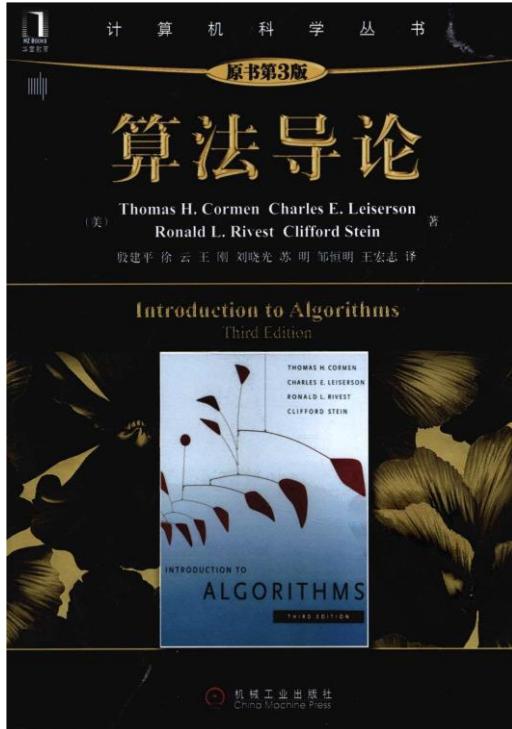
算法设计与分析

苏州大学 计算机科学与技术学院

汪笑宇

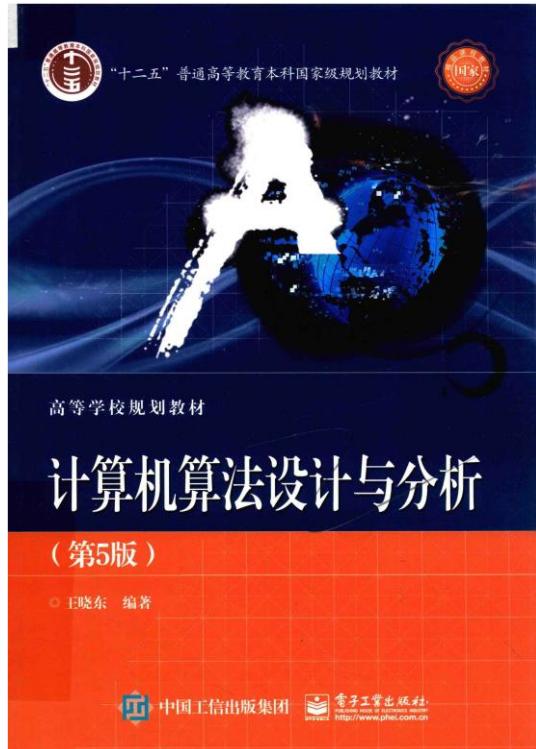
Email: xywang21@suda.edu.cn

教材



《算法导论》(第3版)

参考书目



王晓东 编著
《计算机算法设计与分析》(第5版)
电子工业出版社

课程总览

■本学期课程涉及内容：

- Chapter 1：算法在计算中的作用
- Chapter 2：算法基础
- Chapter 3：函数的渐近增长
- Chapter 34：NP完全性理论
- Chapter 4：分治策略
- Chapter 7：快速排序
- Chapter 9：中位数和顺序统计量
- Chapter 15：动态规划
- Chapter 16：贪心算法
- Chapter 35：近似算法
- 回溯法（参考书目第5章）
- 分支限界法（参考书目第6章）

评分要求

■ 平时20%

■ 期中20%

■ 期末60%

作业严禁抄袭！！！
请勿直接抄写AI生成内容！！！

引入1

■假设有100个砝码外观完全相同，已知其中有一个是劣质的质量稍轻，而其他99个质量相同，如何用一个天平快速得出哪一个是劣质的砝码？

➤想法一：随机挑一个砝码作为基准，放在天平一侧，剩下的砝码依次放在天平另一侧进行称重

- 最坏情况：前98次称重天平都平衡
- 最好情况：第1次称重就天平不平衡

引入1 (续)

想法二：

- 第1次：天平一边50个，重的那一侧排除
- 第2次：天平一边25个，重的那一侧排除
- 第3次：天平一边12个，剩下1个，一样重则剩下的一个劣质；不一样重则重的一侧和剩下的一个排除
- 第4次：天平一边6个，重的那一侧排除
- 第5次：天平一边3个，重的那一侧排除
- 第6次：天平一边1个，剩下1个，一样重则剩下的一个劣质；否则哪边轻哪边劣质

二分法

$$\lfloor \log_2 100 \rfloor = 6$$

引入2

■ 向你常用的大模型助手问一个你关心的问题



文心一言



GPT



通义千问



ChatGLM 3



DeepSeek



ABAB



XVERSE



PaLM 2



Llama 2



360智脑



TigerBot



书生·浦语



豆包



混元



百川



日日新



序列猴子



孟子



Moonshot

假设有100个砝码外观完全相同，已知其中有一个是劣质的质量稍轻，而其他99个质量相同，如何用一个天平快速得出哪一个是劣质的砝码？

引用 6 篇资料作为参考 >

已深度思考(用时6秒) ✓

要快速找出100个砝码中唯一一个质量稍轻的劣质砝码，需采用“**三分法**”策略，即每次将砝码尽可能均分为三组，通过天平称量逐步缩小次品范围。此方法基于天平每次称量可提供三种结果（左重、右重、平衡），最大化信息获取效率。以下是具体操作步骤：

最优策略步骤

1. 第一次称量（缩小范围至34个）

- 将100个砝码分为三组：33个（A组）、33个（B组）、34个（C组）。
- 称量A组与B组：
 - **若平衡**：次品在C组（34个）。
 - **若不平衡**：次品在较轻的一组（如A组轻则次品在A组）。
- 此时次品范围缩小至33或34个砝码。

2. 第二次称量（缩小范围至11-12个）

- 对可疑组（假设为34个）再分三组：11个（D组）、11个（E组）、12个（F组）。
- 称量D组与E组：



关键原理与优化

- **三分法优势：**每次称量将问题规模缩小至约 $1/3$ ，是效率最高的策略。若用二分法（每次分两组），需更多次数（如100个砝码需7次）? 5。

- **分组技巧：**

- 尽量均分三组（如 $100 \rightarrow 33+33+34$ ），若不能整除，使两组数量相同，第三组多1或少1。
- 每次称量需覆盖“最坏情况”，确保任何结果下都能继续缩小范围2 6。

- **称量次数：**

- 根据公式 $3^k \geq N$ (N 为总数, k 为最少次数)， 100 在 $3^4 = 81$ 和 $3^5 = 243$ 之间，故需 **5次**（见下表）3 5。

砝码总数范围	最少称量次数
2~3	1次
4~9	2次
10~27	3次
28~81	4次
82~243	5次

思考

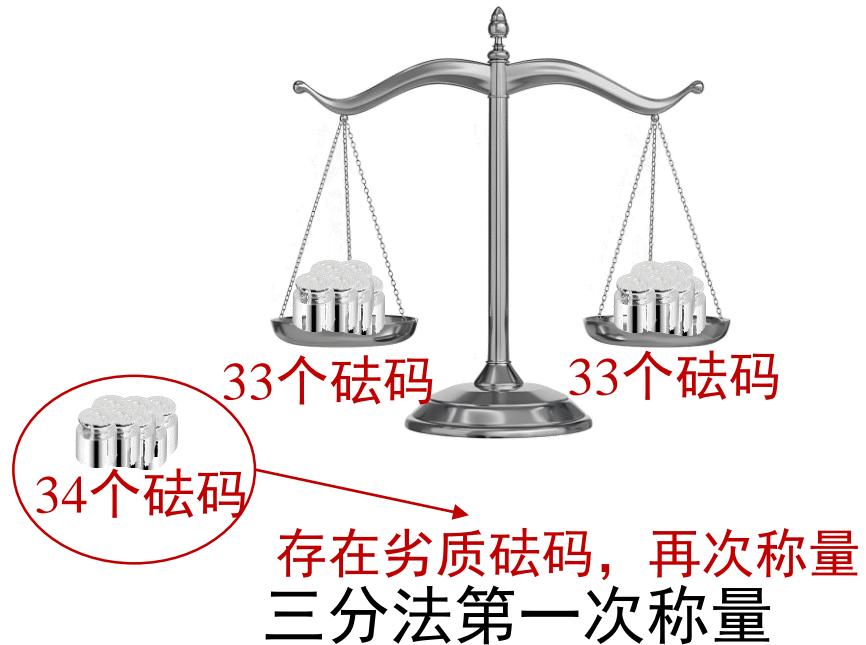
■为什么最坏称量次数二分法向下取整而三分法需要向上取整？

需再次称量的砝码数
 \leq 当前数量的 $1/2$



存在劣质砝码，再次称量
二分法第三次称量

需再次称量的砝码数
 \geq 当前数量的 $1/3$



存在劣质砝码，再次称量
三分法第一次称量

引入2 (续)



松鼠桂鱼？



孔雀舞？

启示

- 正确性
- 算法效率（时间复杂度、空间复杂度）
- 如何设计算法处理特定的问题？（基本策略：
分治法、动态规划、贪心等）

第1章 算法基础

苏州大学 计算机科学与技术学院

汪笑宇

Email: xywang21@suda.edu.cn

本章内容

- 算法定义及基本概念（教材Chapter 1）
- 算法描述（教材Chapter 2）
- 函数增长及渐近记号表示（教材Chapter 3）
- 标准记号与常用函数（教材Chapter 3）
- NP完全性理论（区分并理解P/ NP/ NPC/ NP-Hard 几类问题）（教材Chapter 34）

本章内容

- 算法定义及基本概念（教材Chapter 1）
- 算法描述（教材Chapter 2）
- 函数增长及渐近记号表示（教材Chapter 3）
- 标准记号与常用函数（教材Chapter 3）
- NP完全性理论（区分并理解P/ NP/ NPC/ NP-Hard 几类问题）（教材Chapter 34）

基本概念——算法

■ 非形式定义

➤ 一个算法是任何一个良定义(well-defined)的计算过程，它接收某个值或值的集合作为输入，产生某个值或值的集合作为输出。因此，一个算法是一个计算步骤的序列，这些步骤将输入转化为输出。

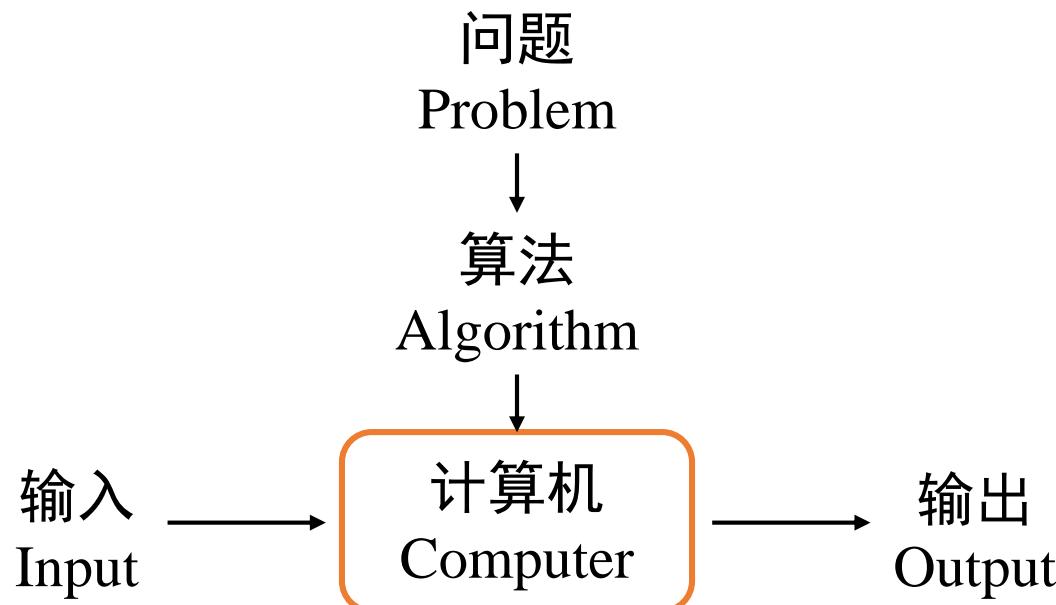


➤ 或者说，算法所描述的计算过程就是怎样达到所期望的I/O关系

基本概念——算法 (Algorithm)

■另一种定义

➤ An **algorithm** is a sequence of **unambiguous** instructions for solving a problem, i.e., for obtaining a required output for any **legitimate** input in a **finite** amount of time.



例：排序问题

■问题描述

- 输入：具有 n 个数的序列 $\langle a_1, a_2, \dots, a_n \rangle$
- 输出：输入序列的一个排列 $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，满足 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

■计算步骤

- 如何达到上述关系

基本概念

- **实例 (Instance)**: 一个问题的**实例**由计算该问题的一个解所需要的**所有输入**所组成
- **正确性**: 若对**每个输入实例**, 算法都以**正确的输出停机**, 则称该算法是**正确的**
 - 不正确的算法: 对某些输入实例不停机
以不正确的输出停机
 - **注**: 不正确的算法只要其错误率可控有时可能是有用的(例如Chapter 31 大素数算法)
- **算法的描述**: 必须精确描述计算过程

算法 vs. 程序

■ 算法 Algorithm ? 程序 Program

- Algorithm (webster.com): A procedure for solving a mathematical problem (as of finding the greatest common divisor) in a finite number of steps that frequently involves repetition of an operation.
- Broadly: a step-step procedure for solving a problem or accomplishing some end especially by a computer.
- Issues: correctness, efficiency (amount of work done and space used), storage (simplicity, clarity), optimality, etc.

算法 vs. 程序 (续)

■ **算法**是指解决问题的一种方法或一个过程，是若干**指令**的**有穷序列**，满足性质：

- **输入**：有外部提供的量作为算法的输入
- **输出**：算法产生至少一个量作为输出
- **确定性**：组成算法的每条指令是清晰、无歧义的
- **有限性**：算法中每条指令的执行次数是有限的，执行每条指令的时间也是有限的

注：正确性是前提

算法 vs. 程序 (续)

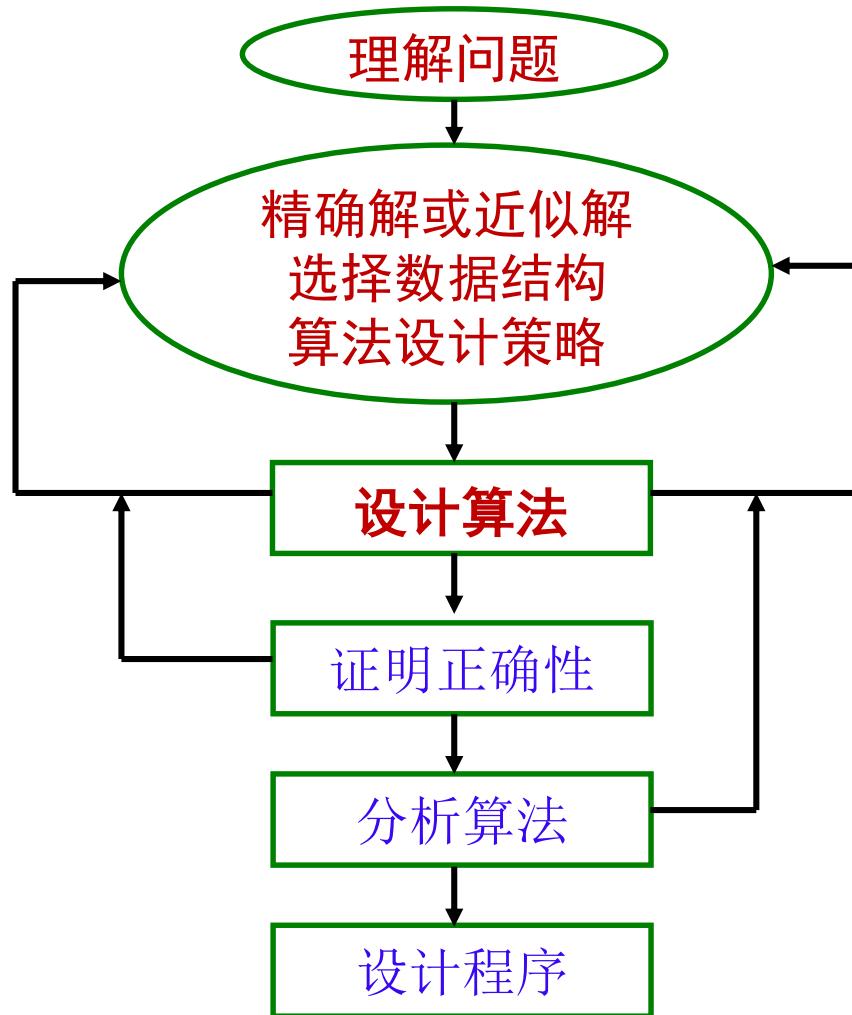
■ 程序

- 程序是算法用某种程序设计语言的具体实现
- 程序可以不满足有限性

例如：操作系统是一个在无限循环中执行的程序，因而不是一个算法。

操作系统的各种任务可看成是单独的问题，每一个问题由操作系统中的一个子程序通过特定的算法来实现。该子程序得到输出结果后便终止。

算法 -> 问题求解



算法解决哪些问题

■ 人类基因工程

- 识别10万个基因，确定30亿个化学碱基对序列

■ 互联网

- 为数据传输寻找好的路由
- 使用搜索引擎快速找到特定信息所在网页

■ 电子商务

- 公钥密码和数字签名

■ 制造业

- 按最有益的方式分配稀有资源

例：求最大公约数

■问题：求 m 和 n 的最大公约数 $\gcd(m, n)$ ，其中， m 和 n 为非负整数且不同时为0

➤例： $\gcd(60, 24) = 12$, $\gcd(8, 0) = 8$

■欧几里得算法 (Euclid's algorithm) 基于以下性质：

$$\gcd(m, n) = \gcd(n, m \bmod n)$$

直到第二个参数为0则停止

➤例： $\gcd(60, 24) = \gcd(24, 12) = \gcd(12, 0) = 12$

欧几里得算法的两种描述

步骤1：若 $n = 0$ ，则返回 m 并终止；

否则执行步骤2

步骤2： m 除以 n ，并将 r 赋值为余数

步骤3：将 n 的值赋给 m 、 r 的值赋给 n ，执行步骤1

Step 1: If $n = 0$, return m and stop;
otherwise go to Step 2.

Step 2: Divide m by n and assign the
value for the remainder to r .

Step 3: Assign the value of n to m and
the value of r to n . Go to Step 1.

Euclid(m, n)

while $n \neq 0$ do

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m

求 $\text{gcd}(m, n)$ 的其他算法

■例1：连续整数测试

Step 1: Assign the value of $\min\{m,n\}$ to t

Step 2: Divide m by t . If the remainder is 0,
go to Step 3; otherwise, go to Step 4

Step 3: Divide n by t . If the remainder is 0,
return t and stop; otherwise, go to Step 4

Step 4: Decrease t by 1 and go to Step 2



正确性?

求 $\gcd(m, n)$ 的其他算法 (续)

■例2：分解质因数

Step 1: Find the prime factorization of m

Step 2: Find the prime factorization of n

Step 3: Find all the common prime factors

Step 4: Compute the product of all the common prime factors and return it as $\gcd(m, n)$



这是一个算法吗?

埃拉托色尼筛选法 (Sieve of Eratosthenes)

Input: Integer $n \geq 2$

Output: List of primes less than or equal to n

for $p \leftarrow 2$ to n do $A[p] \leftarrow p$

for $p \leftarrow 2$ to $\lfloor \sqrt{n} \rfloor$ do

if $A[p] \neq 0$

$j \leftarrow p * p$

while $j \leq n$ do

$A[j] \leftarrow 0$

$j \leftarrow j + p$

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

算法的重要性

■问题：对 $n=10,000,000$ 个整数排序

➤计算机A：每秒执行**百亿**(10^{10})条指令，算法需要 **$2n^2$** 条指令

$$\frac{2 \cdot (10^7)^2 \text{条指令}}{10^{10} \text{条指令/秒}} = 20000 \text{秒} \text{ (多于5.5小时)}$$

➤计算机B：每秒执行**千万**(10^7)条指令，算法需要 **$50nlgn$** 条指令

$$\frac{50 \cdot 10^7 \lg 10^7 \text{条指令}}{10^7 \text{条指令/秒}} \approx 1163 \text{秒} \text{ (少于20分钟)}$$

算法的重要性 (续)

Run time (nanoseconds)	$1.3 N^3$	$10 N^2$	$47 N \log_2 N$	$48 N$
Time to solve a problem of size	1000	1.3 seconds	10 msec	0.4 msec
	10,000	22 minutes	1 second	6 msec
	100,000	15 days	1.7 minutes	78 msec
	million	41 years	2.8 hours	0.94 seconds
	10 million	41 millennia	1.7 weeks	11 seconds
Max size problem solved in one	second	920	10,000	1 million
	minute	3,600	77,000	49 million
	hour	14,000	600,000	2.4 billion
	day	41,000	2.9 million	76 billion
N multiplied by 10, time multiplied by	1,000	100	10+	10

本章内容

- 算法定义及基本概念（教材Chapter 1）
- 算法描述（教材Chapter 2）
- 函数增长及渐近记号表示（教材Chapter 3）
- 标准记号与常用函数（教材Chapter 3）
- NP完全性理论（区分并理解P/ NP/ NPC/ NP-Hard 几类问题）（教材Chapter 34）

伪代码 (Pseudo code)

- 伪代码描述：更简洁地表达算法的本质，忽略细节
- 伪代码中的一些约定（书本p11）

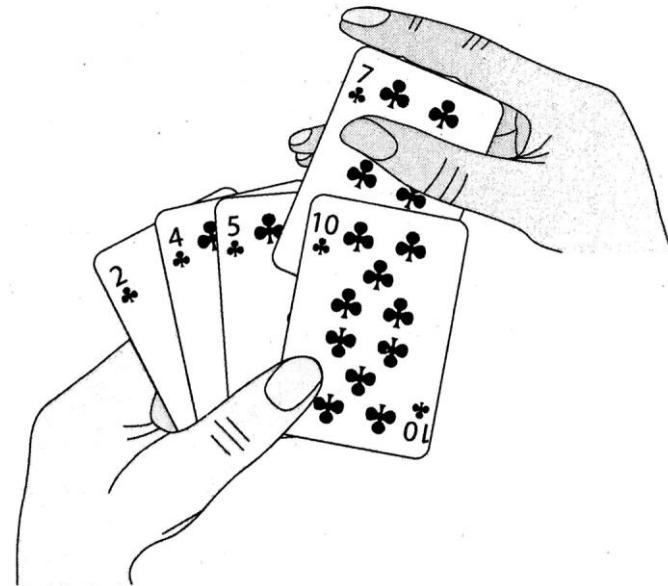
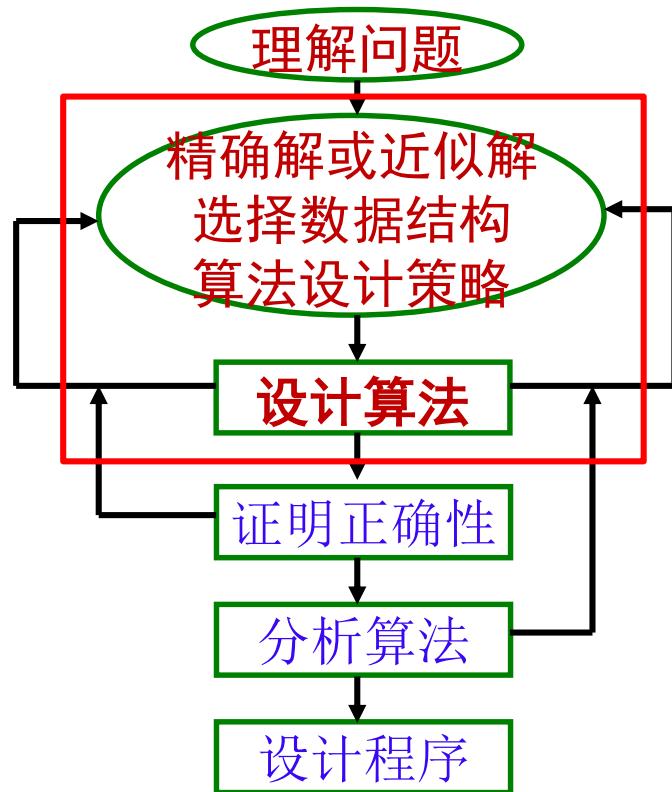
```
INSERTION_SORT(A)
1 for j  $\leftarrow$  2 to A.length do
2   key  $\leftarrow$  A[j]
3   // Insert A[j] into the sorted sequence A[1..j-1]
4   i  $\leftarrow$  j - 1
5   while i > 0 and A[i] > key do
6     A[i+1]  $\leftarrow$  A[i]
7     i  $\leftarrow$  i - 1
8   A[i+1]  $\leftarrow$  key
```

排序问题

- 输入：具有 n 个数的序列 $\langle a_1, a_2, \dots, a_n \rangle$
- 输出：输入序列的一个排列 $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，
满足 $a'_1 \leq a'_2 \leq \dots \leq a'_n$
- 例：
 - 输入： $\langle 5, 2, 4, 6, 1, 3 \rangle$
 - 输出： $\langle 1, 2, 3, 4, 5, 6 \rangle$

排序问题求解：以插入排序为例

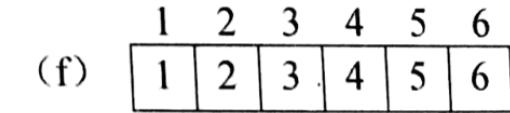
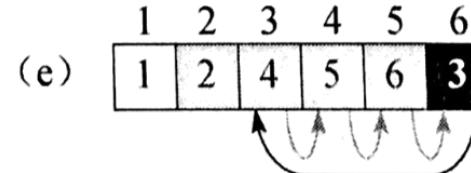
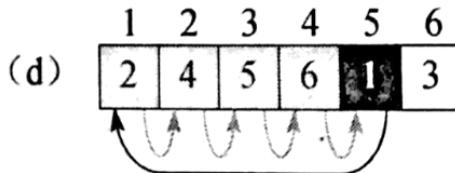
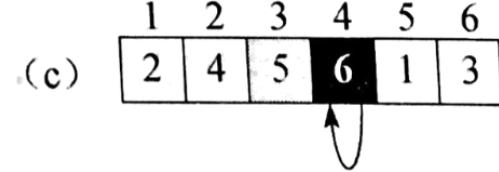
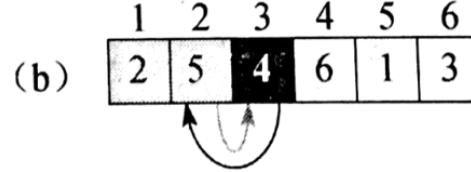
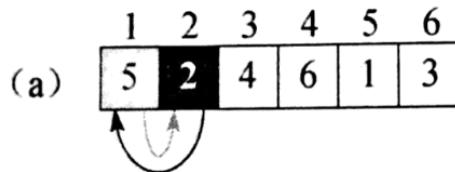
■ 精确解、线性表顺序存储结构、插入排序方法



插入排序执行过程

```
INSERTION_SORT( $A$ )
```

```
1 for  $j \leftarrow 2$  to  $A.length$  do
2    $key \leftarrow A[j]$ 
3   // Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ 
4    $i \leftarrow j - 1$ 
5   while  $i > 0$  and  $A[i] > key$  do
6      $A[i+1] \leftarrow A[i]$ 
7      $i \leftarrow i - 1$ 
8    $A[i+1] \leftarrow key$ 
```



课程回顾

■ 算法是什么？

- 良定义、合法输入、输出、有穷步骤

■ 算法 vs. 程序

- 算法：输入、输出、确定性、有限性（正确性前提）
- 程序：算法的具体实现、可不满足有限性

■ 问题求解步骤

- 理解问题、精确解近似解/数据结构/算法设计策略、设计算法、证明正确性、分析算法、设计程序

■ 算法描述及算法重要性

■ 插入排序分析

■ **循环不变式：**循环体每次执行前后均为真的谓词

➤ 作用：主要用来帮助我们理解算法的正确性

■ **循环不变式性质：**

➤ 初始话：循环的第一次迭代之前，它为真

➤ 保持：如果循环的某次迭代之前它为真，那么下次迭代之前它仍为真

➤ 终止：在循环终止时，不变式为我们提供一个有用的性质，该性质有助于证明算法是正确的

循环不变式 (续)

- 前两条性质与我们熟知的数学归纳法很类似
- 数学归纳法：

- 1、证明当 $n = 1$ 时命题成立。（基本情况）
- 2、假设 $n = m$ 时命题成立，那么可以推导出在 $n = m + 1$ 时命题也成立。（ m 代表任意正整数）（归纳步）

- 循环不变式“初始化” 对应于“基本情况”
- 循环不变式“保持” 对应于“归纳步”
- 区别在于：数学归纳法中的归纳步无限的使用，而
循环不变式中当循环终止时，停止“归纳”

插入排序的正确性证明

```
INSERTION_SORT( $A$ )
1 for  $j \leftarrow 2$  to  $A.length$  do
2    $key \leftarrow A[j]$ 
3   // Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ 
4    $i \leftarrow j - 1$ 
5   while  $i > 0$  and  $A[i] > key$  do
6      $A[i+1] \leftarrow A[i]$ 
7      $i \leftarrow i - 1$ 
8    $A[i+1] \leftarrow key$ 
```

循环不变式：
 $A[1..j-1]$ 中的
元素由原来
 $A[1..j-1]$ 元素
组成，且是从
小到大排序的
有序序列

■**初始化：**首先证明在第一次循环迭代之前（当 $j=2$ 时），循环不变式成立。子数组 $A[1..j-1]$ 仅由单个元素 $A[1]$ 组成，实际上就是 $A[1]$ 中原来的元素。而且该子数组是排序好的（当然很平凡）。这表明第一次循环迭代之前循环不变式成立。

插入排序的正确性证明 (续)

```
INSERTION_SORT( $A$ )
1 for  $j \leftarrow 2$  to  $A.length$  do
2    $key \leftarrow A[j]$ 
3   // Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ 
4    $i \leftarrow j - 1$ 
5   while  $i > 0$  and  $A[i] > key$  do
6      $A[i+1] \leftarrow A[i]$ 
7      $i \leftarrow i - 1$ 
8    $A[i+1] \leftarrow key$ 
```

■ **保持：**非形式化地，for循环体的第4~7行将 $A[j-1]$ 、 $A[j-2]$ 、 $A[j-3]$ 等向右移动一个位置，直到找到 $A[j]$ 的适当位置，第8行将 $A[j]$ 的值插入该位置。这时子数组 $A[1..j]$ 由原来在 $A[1..j]$ 中的元素组成，但已按序排列。那么对for循环的下一次迭代增加 j 将保持循环不变式。

插入排序的正确性证明 (续)

```
INSERTION_SORT( $A$ )
1 for  $j \leftarrow 2$  to  $A.length$  do
2    $key \leftarrow A[j]$ 
3   // Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ 
4    $i \leftarrow j - 1$ 
5   while  $i > 0$  and  $A[i] > key$  do
6      $A[i+1] \leftarrow A[i]$ 
7      $i \leftarrow i - 1$ 
8    $A[i+1] \leftarrow key$ 
```

■ **终止：**导致for循环终止的条件是 $j > A.length = n$ 。因为每次循环迭代 j 增加1，那么必有 $j = n + 1$ 。在循环不变式的表述中将 j 用 $n + 1$ 代替，我们有：子数组 $A[1..n]$ 由原来在 $A[1..n]$ 中的元素组成，但已按序排列。注意到，子数组 $A[1..n]$ 就是整个数组，我们推断出整个数组已排序。因此算法正确。

插入排序的正确性证明 (续)

```
INSERTION_SORT(A)
1 for j  $\leftarrow$  2 to A.length do
2   key  $\leftarrow$  A[j]
3   // Insert A[j] into the sorted sequence A[1..j-1]
4   i  $\leftarrow$  j - 1
5   while i > 0 and A[i] > key do
6     A[i+1]  $\leftarrow$  A[i]
7     i  $\leftarrow$  i - 1
8   A[i+1]  $\leftarrow$  key
```

- 注：第二条性质的一种更形式化的处理要求我们对第5~7行的while循环给出并证明一个循环不变式
- 非形式化的分析来证明第二条性质对外层循环成立
- 思考：第5~7行while循环的循环不变式是什么？

- 目的：分析算法就是估计算法所需资源（时间，空间，通信带宽等）
- 计算模型：单处理器，RAM (Random Access Machine, 随机存取机) 模型，其中指令是顺序执行的，无并发操作
- 涉及的知识基础
 - 离散组合数学、概率论、代数等(分析)
 - 程序设计、数据结构(算法设计)

Analyzing algorithms

- We shall assume a generic one-processor, random-access machine (RAM) model of computation.
 - Instructions are executed one after another, with no concurrent operations.
 - Each time, an instruction of a program is executed as an atom operation. An instruction includes arithmetic operations, logical operations, data movement and control operations.
 - Each such instruction takes a constant amount of time.
 - RAM capacity is large enough.

算法分析 (续)

■时间分析：算法耗费时间与输入规模和实例的构成有关

例：插入排序如同手牌整理，整理速度与纸牌数量和牌本身是否有序相关

➤输入规模：通常用整数表示，取决于被研究的问题

- 例1：数组排序：项数
- 例2：两数相乘：最好的度量是总的位数
- 有时需用两个或多个整数表示输入规模，如图的顶点和边

算法分析 (续)

■时间分析

➤运行时间

- 用基本操作的数目 (执行步数) 来度量；
好处是算法分析独立于机器，即任何基本操作看作是单位时间
- 用更接近实际的计算机上实现的时间来度量；
如RAM模型，不同的指令具有不同的执行时间

两者相差一个常数因子

➤最坏情况运行时间

- 输入规模为 n 时，任何输入的最长运行时间

算法分析 (续)

■时间分析

➤ 为什么要分析算法的**最坏运行时间**? (p15)

- 它是算法对于任何输入的运行时间的上界
- 对于某些算法, 最坏情况常常发生, 如在DB中搜索一个并不存在的记录
- **平均运行时间往往和最坏运行时间相当**, 仅常数因子不同 (存在反例!)

➤ 平均运行时间

- 常常**假定一个给定输入规模的所有输入是等概率的**
- 这种可能并不一定成立, 但可以用**随机化算法**强迫它成立

➤ 有时平均时间和最坏时间不是同数量级, 算法选择依据:
最好、最坏的概率较小时, 尽量选择平均时间较小的算法

算法分析 (续)

■ 最坏情况 (Worst case, 通常)

➤ $T(n)=\text{maximum time}$ of algorithm on any input of size n

■ 平均情况 (Average case, 有时)

➤ $T(n)=\text{expected time}$ of algorithm over all inputs of size n

➤ Need assumption of statistics distribution of inputs

■ 最好情况 (Best case, 假象)

➤ Cheat with a slow algorithm that works fast on some input

插入排序算法分析

	代价	次数
INSERTION_SORT(A)		
1 for $j \leftarrow 2$ to $A.length$ do	c_1	n
2 $key \leftarrow A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$	0	$n-1$
4 $i \leftarrow j - 1$	c_4	$n-1$
5 while $i > 0$ and $A[i] > key$ do	c_5	$\sum_{j=2}^n t_j$
6 $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^n (t_j - 1)$
7 $i \leftarrow i - 1$	c_7	$\sum_{j=2}^n (t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

t_j : 对于值 j , 第5行执行while循环测试的次数

插入排序算法分析 (续)

■ 算法执行时间

$$\begin{aligned} T(n) = & c_1 n + c_2(n - 1) + c_4(n - 1) \\ & + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1) \end{aligned}$$

➤ **最好情况：顺序排列**—— $t_j = 1$

$$\begin{aligned} T(n) &= c_1 n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) \\ &= \Theta(n) \end{aligned}$$

- 执行时间是 n 的线性函数

插入排序算法分析 (续)

■ 算法执行时间

➤ 最坏情况：逆序排列

$$\sum_{j=2}^n t_j = \sum_{j=2}^n j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^n (t_j - 1) = \sum_{j=2}^n (j - 1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) \\ &\quad + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8) \\ &= \Theta(n^2) \end{aligned}$$

- 执行时间是 n 的二次函数

插入排序算法分析 (续)

■ 算法执行时间

- 平均情况：在 $A[1..j-1]$ 中一半大于 $A[j]$ 、一半小于 $A[j]$
- t_j 约为 $j/2$
- $T(n) = \Theta(n^2)$
 - 执行时间是 n 的二次函数

插入排序算法分析 (续)

■ 最坏情况：输入逆序

$$T(n) = \sum_{j=2..n} \Theta(j) = \Theta(n^2)$$

■ 平均情况：所有排列情况等可能出现

$$T(n) = \sum_{j=2..n} \Theta(j/2) = \Theta(n^2)$$

■ 插入排序算法快吗？

正确性证明及算法分析例

- 2-2 (冒泡排序的正确性) 冒泡排序是一种流行但低效的排序算法, 它的作用是反复交换相邻的未按次序排列的元素。

BUBBLESORT(A)

```
1 for  $i = 1$  to  $A.length - 1$ 
2   for  $j = A.length$  downto  $i + 1$ 
3     if  $A[j] < A[j - 1]$ 
4       exchange  $A[j]$  with  $A[j - 1]$ 
```

- a. 假设 A' 表示 BUBBLESORT(A) 的输出。为了证明 BUBBLESORT 正确, 我们必须证明它将终止并且有:

$$A'[1] \leq A'[2] \leq \dots \leq A'[n] \quad (2.3)$$

- 其中 $n = A.length$ 。为了证明 BUBBLESORT 确实完成了排序, 我们还需要证明什么?
- b. 为第 2~4 行的 **for** 循环精确地说明一个循环不变式, 并证明该循环不变式成立。你的证明应该使用本章中给出的循环不变式证明的结构。
 - c. 使用(b)部分证明的循环不变式的终止条件, 为第 1~4 行的 **for** 循环说明一个循环不变式, 该不变式将使你能证明不等式(2.3)。你的证明应该使用本章中给出的循环不变式证明的结构。
 - d. 冒泡排序的最坏情况运行时间是多少? 与插入排序的运行时间相比, 其性能如何?

正确性证明及算法分析例 (续)

2-2 (冒泡排序的正确性) 冒泡排序是一种流行但低效的排序算法, 它的作用是反复交换相邻的未按次序排列的元素。

BUBBLESORT(A)

```
1 for  $i = 1$  to  $A.length - 1$ 
2   for  $j = A.length$  downto  $i + 1$ 
3     if  $A[j] < A[j - 1]$ 
4       exchange  $A[j]$  with  $A[j - 1]$ 
```

a. 假设 A' 表示 BUBBLESORT(A) 的输出。为了证明 BUBBLESORT 正确, 我们必须证明它将终止并且有:

$$A'[1] \leq A'[2] \leq \dots \leq A'[n] \quad (2.3)$$

其中 $n = A.length$ 。为了证明 BUBBLESORT 确实完成了排序, 我们还需要证明什么?

下面两部分将证明不等式(2.3)。

- a. 需证明 A' 中的元素与 A 中的元素相同。因为算法中仅对 A 进行了元素交换操作, 因此输出的 A' 只是 A 进行元素重排的数组

正确性证明及算法分析例 (续)

2-2 (冒泡排序的正确性) 冒泡排序是一种流行但低效的排序算法, 它的作用是反复交换相邻的未按次序排列的元素。

BUBBLESORT(A)

```
1  for  $i = 1$  to  $A.length - 1$ 
2      for  $j = A.length$  downto  $i + 1$ 
3          if  $A[j] < A[j - 1]$ 
4              exchange  $A[j]$  with  $A[j - 1]$ 
```

b. 为第 2~4 行的 for 循环精确地说明一个循环不变式, 并证明该循环不变式成立。你的证明应该使用本章中给出的循环不变式证明的结构。

■ b. 循环不变式: $A[j..n]$ 中最小元素是 $A[j]$

- 初始化: $j=A.length=n$, $A[j..n]$ 仅包含一个元素 $A[j]$, 成立
- 保持: 假设 $j=k \geq i+1$ 时成立, 即 $A[k..n]$ 最小元素是 $A[k]$, 那么当 $j=k-1$ 时 (此时循环执行时 $j=k$, 结束后才是 $j=k-1$), 若 $A[k] < A[k-1]$ 则会执行第 4 行交换操作, 此时 $A[k-1]$ 为 $A[k-1..n]$ 最小元素; 反之若 $A[k]$ 大于等于 $A[k-1]$ 不执行操作, $A[k-1]$ 依旧为 $A[k-1..n]$ 最小元素。因此 $j=k-1 \geq i$ 时, $A[k-1..n]$ 中最小元素是 $A[k-1]$, 成立
- 终止: 循环终止时需执行第 2 行判断, $j=i$ 即 $A[i..n]$ 中最小元素是 $A[i]$

正确性证明及算法分析例 (续)

2-2 (冒泡排序的正确性) 冒泡排序是一种流行但低效的排序算法, 它的作用是反复交换相邻的未按次序排列的元素。

BUBBLESORT(A)

```
1  for  $i = 1$  to  $A.length - 1$ 
2      for  $j = A.length$  downto  $i + 1$ 
3          if  $A[j] < A[j - 1]$ 
4              exchange  $A[j]$  with  $A[j - 1]$ 
```

- c. 使用(b)部分证明的循环不变式的终止条件, 为第 1~4 行的 **for** 循环说明一个循环不变式, 该不变式将使你能证明不等式(2.3)。你的证明应该使用本章中给出的循环不变式证明的结构。
- c. 循环不变式: $A[1..i]$ 是 A 中前 i 个最小元素且从小到大排序

- 初始化: $i=1$, $A[1..i]$ 仅包含 $A[1]$, 成立
- 保持: 假设 $i=k \leq n-1$ 时成立, 即 $A[1] \leq A[2] \leq \dots \leq A[k] \leq \min\{A[k+1], \dots, A[n]\}$ 。则 $i=k+1 \leq n$ 时, 由 b 可知: $A[k+1..n]$ 中最小元素是 $A[k+1]$, 即 $A[k+1]=\min\{A[k+1], \dots, A[n]\}$ 。由此可知 $A[1..k+1]$ 是 A 中前 $k+1$ 个最小元素且从小到大排序, 成立
- 终止: 循环终止时, $i=A.length=n$, $A[1..n]$ 是 A 中前 n 个最小元素且从小到大排序, 即数组 A 元素从小到大排序

正确性证明及算法分析例 (续)

2-2 (冒泡排序的正确性) 冒泡排序是一种流行但低效的排序算法, 它的作用是反复交换相邻的未按次序排列的元素。

BUBBLESORT(A)

```
1  for  $i = 1$  to  $A.length - 1$ 
2      for  $j = A.length$  downto  $i + 1$ 
3          if  $A[j] < A[j - 1]$ 
4              exchange  $A[j]$  with  $A[j - 1]$ 
```

d. 冒泡排序的最坏情况运行时间是多少? 与插入排序的运行时间相比, 其性能如何?

■ d. 代价 执行次数

$$c_1 \quad T_1 = n = \Theta(n)$$

$$c_2 \quad T_2 = \sum_{i=1}^{n-1} (n - i + 1) = \Theta(n^2)$$

$$c_3 \quad T_3 = \sum_{i=1}^{n-1} (n - i) = \Theta(n^2)$$

$$c_4 \quad T_4$$

■ 最坏情况: $T_4 = T_3 = \Theta(n^2)$, 总体为 $\Theta(n^2)$

■ 由于 $T_2 = T_3 = \Theta(n^2)$, 因此冒泡排序不论最好最坏平均情况运行时间都是 $\Theta(n^2)$, 而插入排序最好情况运行时间为 $\Theta(n)$

本章内容

- 算法定义及基本概念（教材Chapter 1）
- 算法描述（教材Chapter 2）
- 函数增长及渐近记号表示（教材Chapter 3）
- 标准记号与常用函数（教材Chapter 3）
- NP完全性理论（区分并理解P/ NP/ NPC/ NP-Hard 几类问题）（教材Chapter 34）

渐近增长

■ 插入排序例子中可以看出：

- 关注最坏情况的运行时间，是输入规模 n 的函数
- 不关注常量
- 不关注低阶项

■ 宏观思想：

- 忽略与机器相关的常量
- 关注 $n \rightarrow \infty$ 时 $T(n)$ 的增长情况

渐近表示法

- 算法的渐近时间定义为一个函数，定义域为自然数集合 $\mathbb{N} = \{0, 1, 2, \dots\}$ (因为自变量 n 表示输入规模)。但有时也将其扩展到实数或限制到自然数的某子集上。

Θ记号 (大Θ表示法)

■ **Θ记号**: 给定一个函数 $g(n)$, $\Theta(g(n))$ 表示以下函数的集合:

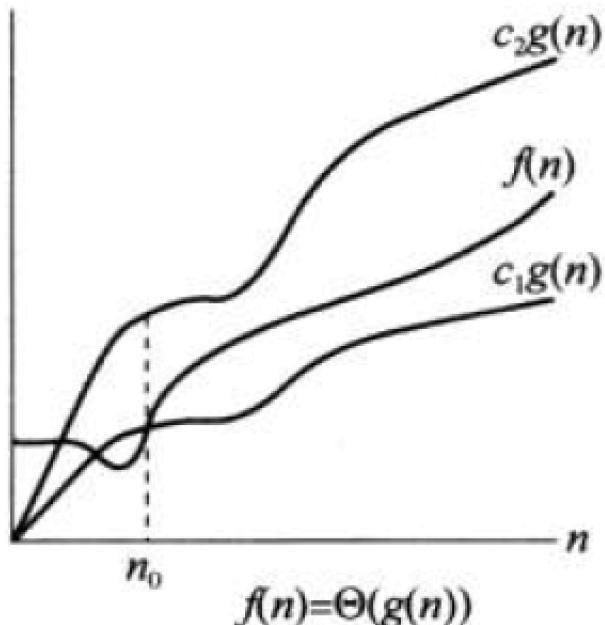
$\Theta(g(n)) = \{f(n): \text{存在正常量} c_1, c_2, n_0, \text{使得对所有} n \geq n_0 \text{有} 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

- 即 $f(n) \in \Theta(g(n))$ 表示存在正常数 c_1, c_2 及足够大的 n , 使得 $f(n)$ 夹在 $c_1 g(n)$ 和 $c_2 g(n)$ 之间
- 通常 $f(n) \in \Theta(g(n))$ 表示为 $f(n) = \Theta(g(n))$
 - 这里的等号表示“属于”关系, 等号两边对调则错误!
- 称 $g(n)$ 是 $f(n)$ 的一个渐近紧确界 (asymptotically tight bound)

Θ 记号 (大 Θ 表示法) (续)

■ **Θ 记号**: 给定一个函数 $g(n)$, $\Theta(g(n))$ 表示以下函数的集合:

$\Theta(g(n)) = \{f(n): \text{存在正常量} c_1, c_2, n_0, \text{使得对所有} n \geq n_0 \text{有} 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$



$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = a, \quad a > 0$$

Θ 记号(大 Θ 表示法)(续)

■**意义**：对所有的 $n \geq n_0$ ，函数 $f(n)$ 在一个常数因子范围内等于 $g(n)$ ，也就是说： $g(n)$ 是 $f(n)$ 的**渐近上界和渐近下界**

➤ $f(n)$ ——算法的计算时间

➤ $g(n)$ ——算法计算时间的数量级

■**注**： $\Theta(g(n))$ 定义中要求 $f(n)$ 和 $g(n)$ 是**渐近非负**的(n 足够大时函数值非负)，否则 $\Theta(g(n))$ 是空集

Θ 记号(大 Θ 表示法)(续)

■例(p27): $T(n)=an^2+bn+c$ ($a>0$) 则 $T(n)=\Theta(n^2)$

取

$$c_1 = \frac{a}{4}, c_2 = \frac{7a}{4}, n_0 = 2 \max\left(\frac{|b|}{a}, \sqrt{\frac{|c|}{a}}\right)$$

则对所有 $n \geq n_0$, $0 \leq c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2$ 成立

■一般来说, 对任意多项式

$$p(n) = \sum_{i=0}^d a_i n^i$$

其中 a_i 为常量且 $a_d > 0$, 我们有 $p(n)=\Theta(n^d)$

Θ 记号 (大 Θ 表示法) (续)

■ 记号 $\Theta(1)$

- 算法运行时间与输入规模 n 无关
- 也可理解为常值函数 $\Theta(n^0)$

■ 渐近符号在公式中替换低阶项用来**简化表达**

- 例： $4n^2+2n+1 = 4n^2+2n+\Theta(1) = 4n^2+\Theta(n) = \Theta(n^2)$

O 记号 (大 O 表示法)

■ **O 记号 (渐近上界)**: 给定一个函数 $g(n)$, $O(g(n))$ 表示以下函数的集合:

$O(g(n)) = \{f(n): \text{存在正常量} c, n_0, \text{使得对所有} n \geq n_0 \text{ 有} 0 \leq f(n) \leq cg(n)\}$

- 即在一个常数因子范围内 $g(n)$ 是 $f(n)$ 的**渐近上界**
- Θ 记号强于 O 记号

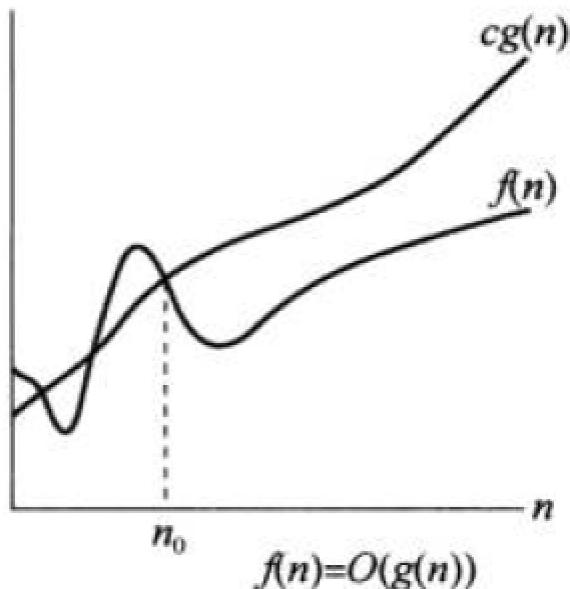
$$\Theta(g(n)) \subseteq O(g(n))$$

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$$

O 记号 (大 O 表示法)

■ **O 记号 (渐近上界)**: 给定一个函数 $g(n)$, $O(g(n))$ 表示以下函数的集合:

$O(g(n)) = \{f(n): \text{存在正常量} c, n_0, \text{使得对所有} n \geq n_0 \text{ 有} 0 \leq f(n) \leq cg(n)\}$



■ $g(n)$ 是 $f(n)$ 的一个渐近上界 (asymptotically upper bound), 限制算法最坏情况运行时间

O 记号 (大 O 表示法) (续)

■注：

- O 记号描述上界，当用于限制算法最坏运行时间时，蕴含着该算法在任意输入上的运行时间都限制于此界
- Θ 记号则不然，一个算法的最坏运行时间是 $\Theta(g(n))$ ，并非蕴含着该算法对每个输入实例的运行时间渐近精确界均为 $\Theta(g(n))$
- 当说算法的运行时间上界是 $O(g(n))$ 往往是指其最坏运行时间，无须修饰语，对那些最好、最坏、平均时间数量级不同者均成立，而 Θ 则要分开表达、加修饰语

■例：插入排序最坏情况运行时间的界 $O(n^2)$ 适用于该算法对每个输入的运行时间，但 $\Theta(n^2)$ 并不适用于所有输入，当输入已排好序时为 $\Theta(n)$

O 记号 (大 O 表示法) (续)

■例1： $T(n)=k_1n^2+k_2n+k_3=O(n^2)$, $k_1>0$

➤由于前面的例子已经说明 $T(n)=\Theta(n^2)$, 因为 $\Theta(n^2)$ 包含于 $O(n^2)$, 所以 $T(n)=O(n^2)$

➤ $k_1n^2+k_2n+k_3 \leq (k_1+|k_2|+|k_3|)n^2$, 因此当 $c=k_1+|k_2|+|k_3|$ 且 $n \geq 1$ 时满足 $0 \leq k_1n^2+k_2n+k_3 \leq cn^2$

■例2： $T(n)=k_1n^2+k_2n+k_3=O(n^3)$, $k_1>0$

➤ $k_1n^2+k_2n+k_3 \leq (k_1+|k_2|+|k_3|)n^3$

O 记号 (大 O 表示法) (续)

■注记：

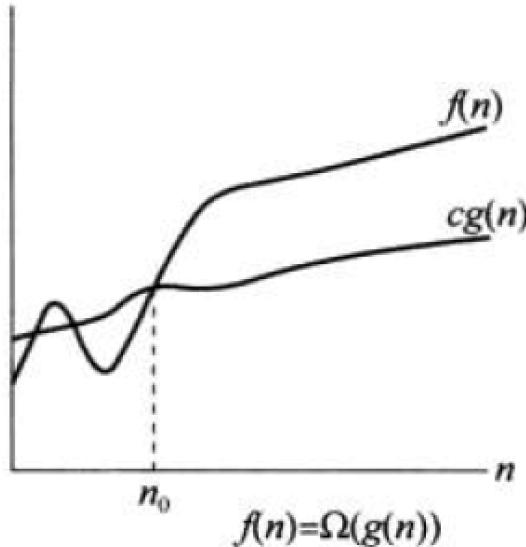
- 当我们说“运行时间是 $O(n^2)$ ”时，通常表示最坏情况运行时间是 $O(n^2)$ ，**最好情况通常更好**
- O 记号使得分析算法变得更加简单
- 记号说明：
 - 通常将 $f(n) \in O(g(n))$ 写作 $f(n)=O(g(n))$
 - 公式中通常也会使用 O 记号，例如 $2n^2+3n+1 = 2n^2+O(n)$ （也就是说 $2n^2+3n+1 = 2n^2+f(n)$ ，其中 $f(n)=O(n)$ ）
 - $O(1)$ 表示常数时间，与输入规模无关

Ω 记号 (大 Ω 表示法)

■ **Ω 记号 (渐近下界)**: 给定一个函数 $g(n)$, $\Omega(g(n))$ 表示以下函数的集合:

$\Omega(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{使得对所有 } n \geq n_0 \text{ 有 } 0 \leq cg(n) \leq f(n)\}$

➤ 即在一个常数因子范围内 $g(n)$ 是 $f(n)$ 的渐近下界



渐近表示法 (续)

■ **定理 3.1** (p28) 对任意两个函数 $f(n)$ 和 $g(n)$, 我们有 $f(n)=\Theta(g(n))$, 当且仅当 $f(n)=O(g(n))$ 且 $f(n)=\Omega(g(n))$ 。

➤ 即 $g(n)$ 是 $f(n)$ 的渐近确界当且仅当 $g(n)$ 是 $f(n)$ 的渐近上界和渐近下界

■ 当 Ω 用来界定一个算法的**最好情况**下的运行时间时, 蕴含着该算法在任意输入上的运行时间都限制于此界

➤ 例: 插入排序的下界是 $\Omega(n)$, 其对任何实例成立(即插入排序的最好运行时间是 $\Omega(n)$)

$$n = \Omega(n), n^2 = \Omega(n)$$

o 记号 (小 o 表示法)

■ o 记号 (渐近非紧上界): 给定一个函数 $g(n)$,
 $o(g(n))$ 表示以下函数的集合:

$$o(g(n)) = \{f(n): \text{对任意常数 } c > 0, \text{ 存在常数 } n_0 > 0, \\ \text{使得对所有 } n \geq n_0 \text{ 有 } 0 \leq f(n) < cg(n)\}$$

➤ O 记号表示的渐近上界可以是渐近紧致的, 也可以是渐近非紧界

- 例如: $5n^2 = O(n^2)$, $5n = O(n^2)$

➤ o 记号表示渐近非紧上界 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

- 例如: $5n = o(n^2)$, $5n^2 \neq o(n^2)$

➤ $f(n)$ 和 $g(n)$ 数量级不同

ω 记号 (小 ω 表示法)

■ **ω 记号 (渐近非紧下界)**: 给定一个函数 $g(n)$,
 $\omega(g(n))$ 表示以下函数的集合:

$\omega(g(n)) = \{f(n): \text{对任意常数} c > 0, \text{存在常数} n_0 > 0,$
使得对所有 $n \geq n_0$ 有 $0 \leq cg(n) < f(n)\}$

➤ Ω 记号表示的渐近下界可以是渐近紧致的, 也可以是
渐近非紧界

- 例如: $5n^2 = \Omega(n^2)$, $5n^3 = \Omega(n^2)$

➤ ω 记号表示渐近非紧上界 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$

- 例如: $5n^3 = \omega(n^2)$, $5n^2 \neq \omega(n^2)$

➤ $f(n)$ 和 $g(n)$ 数量级不同, $f(n)$ 数量级更大

渐近表示法总结

■ $f(n)=\Theta(g(n))$ 渐近紧确界

$\Theta(g(n)) = \{f(n): \text{存在正常量 } c_1, c_2, n_0, \text{使得对所有 } n \geq n_0 \text{ 有 } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

■ $f(n)=O(g(n))$ 渐近上界

$O(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{使得对所有 } n \geq n_0 \text{ 有 } 0 \leq f(n) \leq c g(n)\}$

■ $f(n)=\Omega(g(n))$ 渐近下界

$\Omega(g(n)) = \{f(n): \text{存在正常量 } c, n_0, \text{使得对所有 } n \geq n_0 \text{ 有 } 0 \leq c g(n) \leq f(n)\}$

渐近表示法总结 (续)

■ $f(n)=o(g(n))$ 渐近非紧上界

$o(g(n)) = \{f(n): \text{对任意常数} c>0, \text{存在常数} n_0 > 0, \text{使得对所有} n \geq n_0 \text{有} 0 \leq f(n) < cg(n)\}$

■ $f(n)=\omega(g(n))$ 渐近非紧下界

$\omega(g(n)) = \{f(n): \text{对任意常数} c>0, \text{存在常数} n_0 > 0, \text{使得对所有} n \geq n_0 \text{有} 0 \leq cg(n) < f(n)\}$

$$\blacksquare \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = o(g(n)), f(n) = O(g(n)) \\ a, a > 0 & f(n) = \Theta(g(n)), f(n) = O(g(n)), f(n) = \Omega(g(n)) \\ \infty & f(n) = \omega(g(n)), f(n) = \Omega(g(n)) \end{cases}$$